1
```
#Initialisation
import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import simps
from scipy import signal as sg
from scipy.interpolate import RectBivariateSpline as ReBiSpline
from numpy import ma
from matplotlib import colors, ticker, cm
from random import choice
import scipy.ndimage.filters as filters
import scipy.ndimage.morphology as morphology
import timeit
import math
# import cv2
from PIL import Image
%matplotlib inline
```

2
```
# Read grids from image
im = Image.open("cc_fin_new_hill.bmp")
Base = np.array(im)
```

3
```
# Define internal quantities and variables
scale = 0.25  # m per pixel
Nx = Base[:,0,0].size  # N appears to be resolution
Ny = Base[0,:,0].size  # Nx,Ny is size, Nz is RGB level
xmin=-scale*0.5*(Nx-1)
xmax=scale*0.5*(Nx-1)
ymin=-scale*0.5*(Ny-1)
ymax=scale*0.5*(Ny-1)
x = np.linspace(xmin, xmax, Nx) # This is defining the axes and full space
y = np.linspace(ymin, ymax, Ny)
Y, X= np.meshgrid(y, x)
TrailPotential = np.zeros((Nx,Ny))
DestinationPotential=np.zeros((Nx,Ny))
Weight=np.zeros((Nx,Ny))  # Create gradient to sit on Nx, Ny
intens=np.zeros((Nx,Ny))
q_alpha=np.zeros((Nx,Ny))
expdist=np.zeros((2*Nx-1,2*Ny-1))
dest=np.zeros((2))
start=np.zeros((2))
grad=np.zeros((2,Nx,Ny))
vel=np.asarray([0.,0.])
pos=np.asarray([0.,0.])
#desdirx=ReBiSpline(x,y,grad[0,:,:],s=2)
#desdiry=ReBiSpline(x,y,grad[1,:,:],s=2)
intens[:]=0.

#print(route)
#parameters
t_track=50. # Track decay time - after 50 walkers ignore a trail, it decays by 1/e
dt=0.1  # dt per time step, continuous markings every dt metres
dvel=1. # desired walker velocity in m/s
tau=5.
isigma=1./2. # trail potential
conv_thresh=10.e-4
precision=1.**2 #distance to target.
eps=0.025 #random motion contribution, same for all
p_alpha = 0.3 # value of persistence
prevDir = 0 # record of previous angle of direction
theta_max = 0.05  # maximum permissible ascent angle
```

4
```
class Slope:
    def __init__(self, posx, posy):
```

```
        xdir = gradx[posx][posy] # horizontal slope component at pos
        ydir = grady[posx][posy] # vertical slope component at pos
        self.xdir = xdir
        self.ydir = ydir
        self.mod = math.sqrt(xdir**2 + ydir**2)  # modulus of the gradient
        if self.xdir == 0: # 0 exception for gradient singnularity
            if self.ydir > 0:
                self.angle = math.pi/2
            elif self.ydir < 0:
                self.angle = -math.pi/2
        else:
            self.angle = math.atan(self.ydir/self.xdir)  # direction of the gradient vector  # np.a

    def forbidden_angle(self, curDir):
        grad_max = math.tan(theta_max)
        # print(self.mod)
        if self.mod < grad_max:
            # Ignore gradient processing if maximum gradient is not exceeded anywhere
            return curDir
        else:
            curDir = persistence(curDir) # Apply the persistence (step 7)
            thetaSlope = math.atan(self.mod)  # angle of the slope relative from ground to the heig
            thetaForbMin = self.angle - (math.pi/2 - math.asin(math.tan(theta_max)/math.tan(thetaSl
            # thetaForbMin = self.angle - math.acos(self.mod/grad_max) # Minimum forbidden angle
            thetaForbMax = self.angle + (math.pi/2 - math.asin(math.tan(theta_max)/math.tan(thetaSl
            # thetaForbMax = self.angle + math.acos(self.mod/grad_max) # Minimum forbidden angle
            if (curDir > thetaForbMin) and (curDir <= self.angle):  # current direction within left
                curDir = thetaForbMin
                return curDir
            elif (curDir < thetaForbMax) and (curDir > self.angle):  # current direction within rig
                curDir = thetaForbMax
                return curDir
            else:  # current direction outside of the forbidden zone
                pass # curDir remains unchanged
                return curDir
```

```
5   ##Set up map
    #Create blank arrays for map
    z = np.zeros((Nx,Ny))
    g_max=np.zeros((Nx,Ny)) # empty matrix
    g_nat=np.zeros((Nx,Ny))
    g_grad=np.zeros((Nx,Ny))

    g_nat=np.maximum(np.ones_like(g_nat),np.float64(Base[:,:,0])) # red channel, np.ones_like() sets mi
    g_max=np.maximum(np.ones_like(g_max),np.float64(Base[:,:,1])) # green channel
    g_height=np.maximum(np.ones_like(g_grad),np.float64(Base[:,:,2])) # blue channel
    z=g_nat

    # Trails (start and end point) For current Map, coordinates in metres, centre of image = (0,0)

    # single possible path
    route=np.array([[24.,-9.75],[-24.,9.75]]),

    # commented out for single path

    # route=np.array([[[-2.5,14.],[24.,-9.75]],
    #                 [[-2.5,14.],[24.,2.5]],
    #                 [[-2.5,14.],[-24.,9.75]],
    #                 [[24.,-9.75],[-2.75,14.]],
    #                 [[24.,-9.75],[-24.,9.75]],
    #                 [[24.,2.5],[-2.75,14.]],
    #                 [[24.,2.5],[-24.,9.75]],
    #                 [[-24.,10.],[-2.75,14.]],
    #                 [[-24.,10.],[24.,-9.75]],
```

```
#                 [[-24.,10.],[24.,2.5]]])
```

```
#Setup weight matrix, here trapezoid rule.
Weight[:,:]=1
Weight[1:-1,:]=2
Weight[:,1:-1]=2
Weight[1:-1,1:-1]=4
Weight*=0.25*((x[-1]-x[0])/(Nx-1))*((y[-1]-y[0])/(Ny-1))
#0.25*((x[-1]-x[0])/(N-1))*((y[-1]-y[0])/(N-1))
#np.exp(-np.sqrt((x[:,None]-x[N/2])**2+(y[None,:]-y[N/2])**2))*z[:,:]
```

```
# Setup distance matrix
for xi in range(1,Nx+1):
    for yi in range(1,Ny+1):

        expdist[xi-1,yi-1]=np.exp(-isigma*np.sqrt((x[Nx-xi]-xmin)**2+(y[Ny-yi]-ymin)**2))
        expdist[-xi,-yi]  = expdist[xi-1,yi-1]
        expdist[-xi,yi-1] = expdist[xi-1,yi-1]
        expdist[xi-1,-yi] = expdist[xi-1,yi-1]

# find index range > conv_thresh
subexpdist=expdist[(expdist>conv_thresh).any(1)]
subexpdist=subexpdist[:, np.any(subexpdist>conv_thresh, axis=0)]
#subexpdist=subexpdist[:,np.any(subexpdist>conv_thresh, axis=0)]
#expdist[subexpdist]=0.
subexpdist.shape
#expdist
#subexpdist
```

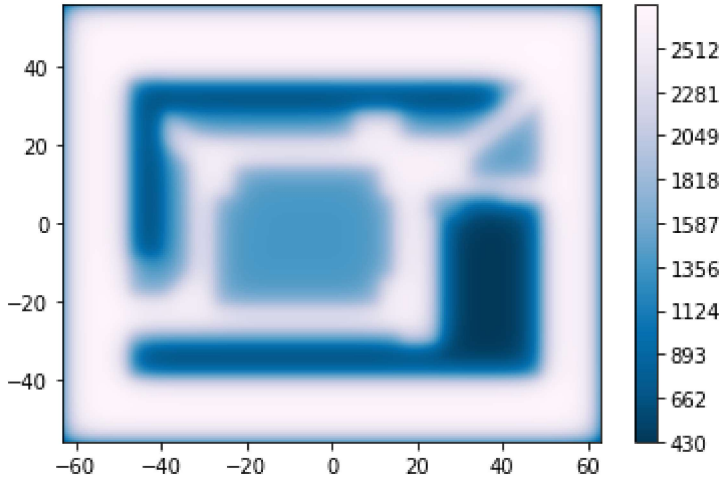
```
(111, 111)
```

```
def calc_tr_new():
    TrailPotential[:,:]=sg.convolve2d(z[:,:]*Weight[:,:],subexpdist[:,:],mode="same")  # 2D convolu
```

```
timeit.timeit(calc_tr_new,number=1)
```

```
8.616709199999999
```

```
# Defines a Plot to show the smoothing of the supplied map to represent the respective potentials o
# the potentials, the more attractive the ground is to the walker

cs = plt.contourf(X, Y, TrailPotential, levels=np.linspace(TrailPotential.min(),TrailPotential.max(
cbar = plt.colorbar()

#plt.scatter(track[0:1999,0],track[0:1999,1])
plt.show()
```



11
```
#set up walker
def set_up_walker(route_id):
    global vel,pos,track,intens,dest,start,route
    #start
    # start=np.array(route[route_id,0,:])  # commented for simplicity
    start = np.array([24.,-9.75])  # temporary one route
    dest = np.array([-24.,9.75])  # temporary one route
    #dest=(random.choice(ends))
    # dest=np.array(route[route_id,1,:]) # commented for simplicity
    vel=np.array([0.,0.])
    pos=np.array(start)
    #print (pos)
    track=np.zeros((2000,2))
    #track[0,:]=pos[:]
```

12
```
#Calculate gradients eq 19
#Trail gradient
def setup_potentials():
    global grad,desdirx,desdiry,dest
    grad=0.003*np.array(np.gradient(TrailPotential))
    #grad=0.002*np.array(np.gradient(TrailPotential)) ORIGINAL

    #print (dest)
    #Destination potential
    DestinationPotential=-np.sqrt((dest[0]-x[:,None])**2+(dest[1]-y[None,:])**2)
    #Combine gradients
    grad+=np.array(np.gradient(DestinationPotential)[:])
    #Normalise
    #grad[:,:,:]/=(np.sqrt(grad[0,:,:]**2+grad[1,:,:]**2))
    desdirx=ReBiSpline(x,y,grad[0,:,:],s=2) # gradeint plus magnitude, Spline approximation over a
    desdiry=ReBiSpline(x,y,grad[1,:,:],s=2)
    # angle array, arctans of gradient components, rebislpine
    # continuous desired angles permissible
        # 2pi periodic system aware
```

13
```
# #Plot the direction
# scgrad=np.arctan2(grad[1],grad[0])
# levels = np.linspace(-np.pi, np.pi, 360)
# cs = plt.contourf(X, Y,scgrad, levels=levels,cmap='hsv')

# cbar = plt.colorbar()
# # ERROR # plt.scatter(track[0:1999,0],track[0:1999,1])
# #plt.scatter(start, dest)
# print(start)
# print(dest)
# plt.show()
```

14
```
def persistence(curDir):
        global prevDir
        # Apply persistence of direction formula (Gilks equation 6) - weighted average of the movem
        gamma = (p_alpha * prevDir) + (1 - p_alpha)*curDir  # previous direction as estimation
        return gamma
```

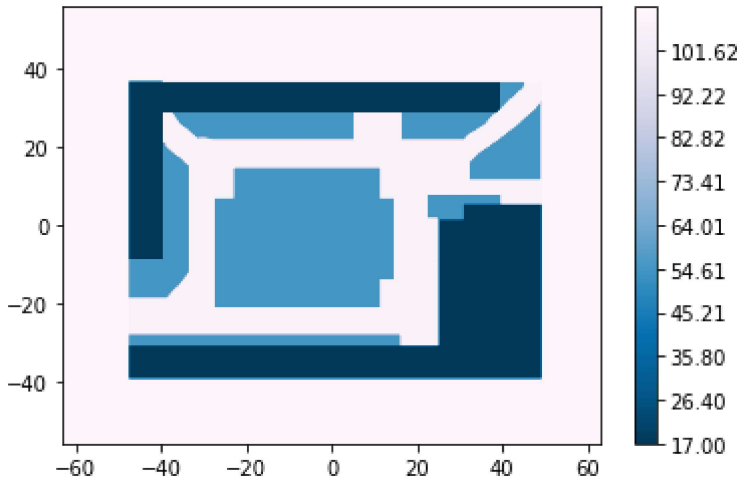
15
```
def calc_path():
    global pos,vel,intens,track,dest,dvel,tau, prevDir, desdirx, desdiry
    i=0
    hist=10
    samp=10
    avpos=np.zeros((2,hist))
    #Setup While loop to run until either the walker reaches the destination or the walker has pass
    #attempt to get there
    while (np.dot(pos-dest,pos-dest)>precision and i<2000):
        #set the postiion of the walker on its first then subsequent cycles
        #conditional logic saying to update the average position of the walker every 10 iterations
        #if (i%samp==0): avpos[:,(i%hist)//samp]=pos[:] #ORIGINAL
        if (i%samp==0):
            avpos[:,(i%(hist*samp))//samp]=pos[:]
        #print((i%hist)//samp)
        # print(avpos)
        gradmagnitude=max(0.0001,np.sqrt(desdirx(pos[0],pos[1])**2+desdiry(pos[0],pos[1])**2))
        xi=np.array(np.random.normal(0,1,2))
        # Equation 6 in Helbing, differential in position, eliminised velocity decay components
        # gradmagnitude makes sure it is normalised, desdir not normalised
        pos[0]+= dt *(dvel * desdirx(pos[0],pos[1])/gradmagnitude +np.sqrt(2.*eps/tau)*xi[0])  # x-
        pos[1]+= dt *(dvel * desdiry(pos[0],pos[1])/gradmagnitude +np.sqrt(2.*eps/tau)*xi[1])  # y-
#       posGrad = math.degree(math.atan(pos[0]/pos[1]) # future position
        curDir = math.atan(desdiry(pos[0],pos[1])/desdirx(pos[0],pos[1]))
        curDir = (curDir * p_alpha) + (prevDir*(1-p_alpha))
        curGradient = Slope(round(pos[0]),round(pos[1]))  # set the current gradient to the Slope c
        curDir = curGradient.forbidden_angle(curDir) # apply forbidden angle rule
        print('pos[0]=' + str(pos[0]) + '  pos[1] =' + str(pos[1]))
        if prevDir != curDir:  # the position angle has changed, recalculate the path position ???
            pos[0] += dt*(dvel*math.cos(curDir))
            pos[1] += dt*(dvel*math.sin(curDir))
        prevDir = curDir # applying a back trace of the previous direction# Calculate current facin
        # pos+=dt*vel
        #vel[0]+=-1/tau*vel[0] + (dvel/tau)*desdirx(pos[0],pos[1])/gradmagnitude+np.sqrt(2.*eps/tau
        #vel[1]+=-1/tau*vel[1] + (dvel/tau)*desdiry(pos[0],pos[1])/gradmagnitude+np.sqrt(2.*eps/tau

        #Set the current position of the walker into the trakc array for the current iteration
        track[i,:]=pos[:]
        intens[int((pos[0]-xmin)*(Nx-1)/(xmax-xmin)),int((pos[1]-ymin)*(Ny-1)/(ymax-ymin))]+=1.
        i+=1
        if (i%(hist*samp)==0):
            meanpos=np.mean(avpos,axis=1)
            if (np.dot(pos-meanpos,pos-meanpos)<precision):
                print ("Stalled progress ",pos,meanpos,vel, dest)
                break
    if (i==2000): print ("Missed goal ",dest,pos)
    return i
#stopping condition
```

16
```
# Calculate Q_alpha (strength of markings) eq 15
def update_ground():
    global q_alpha,intens,z,g_max,t_track,g_nat
    q_alpha=intens*(1.-z/g_max)
    # Time evolution of ground potential
    #zdiff=(1./t_track)*(g_nat-z)+q_alpha
    z+=(1./t_track)*(g_nat-z)+q_alpha
    #cs = plt.contourf(X, Y, zdiff, cmap=cm.PuBu_r)
    #cbar = plt.colorbar()
```

```
        #plt.show
        #z[140:160,45:75]


17  def plot_path():
        plt.contourf(X, Y, z, levels=np.linspace(z.min(),z.max(),1000),cmap='PuBu_r')
        plt.colorbar()
        #plt.scatter(track[0:1999,0],track[0:1999,1],1)
        plt.show(block=False)


18  tau = 5.


19  g_slope = np.gradient(g_height,1)
    grady = g_slope[0] # vertical slope compontent
    gradx = g_slope[1] # horizontal slope component
    for i in range(0,1):
        calc_tr_new()
        intens[:]=0.
        for j in range(0,1):
            set_up_walker(np.random.randint(0,len(route)))
            setup_potentials()
            #calc_path()
            print(i, start," -> ", dest, pos, calc_path())
        update_ground()
        #plot_path()

    pos[0]=23.91226832474229  pos[1] =-9.728657785536042
    pos[0]=23.92039890851604  pos[1] =-9.723428377260388
    pos[0]=23.918183338983315  pos[1] =-9.709454001976912
    pos[0]=23.92869747414754  pos[1] =-9.70639637807526
    pos[0]=23.915531633609994  pos[1] =-9.710252757238525
    pos[0]=23.90426052191051  pos[1] =-9.72103040257062
    pos[0]=23.90795906294637  pos[1] =-9.728536290026316
    pos[0]=23.922563708043004  pos[1] =-9.730164017102407
    pos[0]=23.9236618667315  pos[1] =-9.724800216584907
    pos[0]=23.922708198098423  pos[1] =-9.719934257747056
    pos[0]=23.924506656493236  pos[1] =-9.720391207149472
    pos[0]=23.936098820791702  pos[1] =-9.727406446598895
    pos[0]=23.938759182455954  pos[1] =-9.744084344850956
    pos[0]=23.944093870472912  pos[1] =-9.743715286764624
    pos[0]=23.939983015931922  pos[1] =-9.746906244447104
    pos[0]=23.935479753325932  pos[1] =-9.755381596780676
    pos[0]=23.93716733798395  pos[1] =-9.773433527197332
    pos[0]=23.92695740011885  pos[1] =-9.76215364262829
    pos[0]=23.917761854568845  pos[1] =-9.776558516140105
    pos[0]=23.929603957770333  pos[1] =-9.76958918135202
    pos[0]=23.92675107086262  pos[1] =-9.77400317104969
    pos[0]=23.936023573333745  pos[1] =-9.788294049294366
    pos[0]=23.93666064470972  pos[1] =-9.795827395460456
    pos[0]=23.930233850448186  pos[1] =-9.789211995631396
    pos[0]=23.93009441557335  pos[1] =-9.785136474335287
    pos[0]=23.922094827966486  pos[1] =-9.789002388737918
    pos[0]=23.94911303116429  pos[1] =-9.774564157187886
    pos[0]=23.95003419922468  pos[1] =-9.77567817604454
    pos[0]=23.941647878129505  pos[1] =-9.768077175933394
    pos[0]=23.942083154669767  pos[1] =-9.763793823007655
    pos[0]=23.94912588845084  pos[1] =-9.77029540808567
    pos[0]=23.955001860315406  pos[1] =-9.774293062495232
    pos[0]=23.95413627195544  pos[1] =-9.791622808786503
    pos[0]=23.949440950699802  pos[1] =-9.786251154466516
    pos[0]=23.95163462078163  pos[1] =-9.800171273364603
    pos[0]=23.96341314334096  pos[1] =-9.793446095537485
    pos[0]=23.978967637668827  pos[1] =-9.787801240301974
    pos[0]=23.974295399918283  pos[1] =-9.790640981709283
    pos[0]=23.969869372999607  pos[1] =-9.805951123734156
    pos[0]=23.976107037484866  pos[1] =-9.804677850511048
```

```
pos[0]=23.97556565121071  pos[1] =-9.80630366994231
pos[0]=23.979473896558737  pos[1] =-9.801179727467497
pos[0]=23.991492772472654  pos[1] =-9.821101690822374
pos[0]=23.988597125265784  pos[1] =-9.808065897250332
pos[0]=23.993561585873397  pos[1] =-9.81680926924049
pos[0]=23.994340222806645  pos[1] =-9.810500371290457
pos[0]=24.001542363723914  pos[1] =-9.815546396907152
pos[0]=24.007083778152523  pos[1] =-9.83374357234124
pos[0]=23.999315093709956  pos[1] =-9.841933275465683
pos[0]=23.984398549251804  pos[1] =-9.854629169044268
pos[0]=23.982274013529707  pos[1] =-9.832304180363664
pos[0]=23.984079325446615  pos[1] =-9.816129845037663
pos[0]=23.98440293397414  pos[1] =-9.824183014174936
pos[0]=23.98216356707006  pos[1] =-9.829442900070257
pos[0]=23.980899682272113  pos[1] =-9.813477837202425
pos[0]=23.999457471917683  pos[1] =-9.829841741323419
pos[0]=24.010363730655136  pos[1] =-9.850577640518045
pos[0]=24.006200303128846  pos[1] =-9.851345896997385
pos[0]=23.99672453637911  pos[1] =-9.862842984044036
pos[0]=23.98148725623595  pos[1] =-9.83705862815992
pos[0]=23.986292511712307  pos[1] =-9.83942855255139
pos[0]=23.997961079596458  pos[1] =-9.837568889036145
pos[0]=24.010508798054584  pos[1] =-9.831523240611618
pos[0]=24.002216365324003  pos[1] =-9.830859754586932
pos[0]=23.99119706146036  pos[1] =-9.83652412943711
pos[0]=24.010626312171016  pos[1] =-9.824298089835729
pos[0]=24.025089383045533  pos[1] =-9.81291114953455
pos[0]=24.04041101607091  pos[1] =-9.795219496273441
pos[0]=24.044085365765014  pos[1] =-9.814429977166986
pos[0]=24.040263954966562  pos[1] =-9.807934026785667
pos[0]=24.047547721632096  pos[1] =-9.8114210380786
pos[0]=24.04433073877863  pos[1] =-9.808334738516642
pos[0]=24.034572329978964  pos[1] =-9.822508100695307
pos[0]=24.027457719601607  pos[1] =-9.83218838484055
pos[0]=24.032481637380684  pos[1] =-9.846089923475356
pos[0]=24.02808465417131  pos[1] =-9.848164190048031
pos[0]=24.02837510914173  pos[1] =-9.856716104152632
pos[0]=24.034120541873122  pos[1] =-9.838901777873883
pos[0]=24.04687701611212  pos[1] =-9.83855586170555
pos[0]=24.04654634727956  pos[1] =-9.82675282540484
pos[0]=24.048575060165007  pos[1] =-9.824366073753618
pos[0]=24.060137904722236  pos[1] =-9.835013231756724
pos[0]=24.05141872844184  pos[1] =-9.821508512354239
pos[0]=24.067427295826267  pos[1] =-9.81129359235405
pos[0]=24.04875360515916  pos[1] =-9.79645099826
pos[0]=24.04147351710762  pos[1] =-9.788394603140375
pos[0]=24.038863885930283  pos[1] =-9.774310897884256
pos[0]=24.03433642790956  pos[1] =-9.772479073182323
pos[0]=24.041085812170365  pos[1] =-9.768795531556172
pos[0]=24.041243501859146  pos[1] =-9.760765898933776
pos[0]=24.028660425705255  pos[1] =-9.762193638307481
pos[0]=24.039739162645493  pos[1] =-9.762329931288692
pos[0]=24.037462465630508  pos[1] =-9.758616300490567
pos[0]=24.03338353070469  pos[1] =-9.771609396889756
pos[0]=24.028509024558076  pos[1] =-9.764489083174182
pos[0]=24.03418171472458  pos[1] =-9.76847898514896
pos[0]=24.046348822765047  pos[1] =-9.777728023510027
pos[0]=24.03232286297207  pos[1] =-9.788310895805086
pos[0]=24.041612133366577  pos[1] =-9.781750912329409
pos[0]=24.03762658432986  pos[1] =-9.783198858542775
Stalled progress  [24.13565525 -9.8029569 ] [24.07464678 -9.80740998] [0. 0.] [-24.     9.75]
0 [24.   -9.75]  ->  [-24.     9.75] [24.13565525 -9.8029569 ] 100
```

```
20  plot_path()
```



21
```
for i in range(0,Nx):
    for j in range(0,Ny):
        if (np.isnan(z[i,j])):
            print (i,j,g_max[i,j],Base[i,j,0])
```

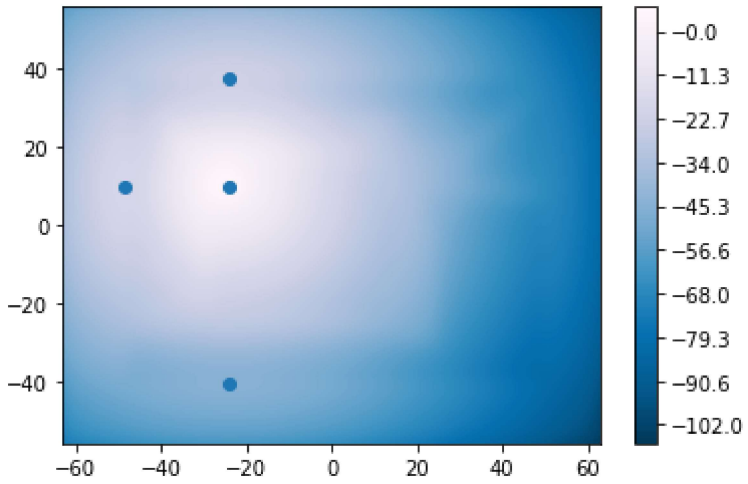
22
```
def detect_local_maxima(arr):
    # https://stackoverflow.com/questions/3684484/peak-detection-in-a-2d-array/3689710#3689710
    """
    Takes an array and detects the troughs using the local maximum filter.
    Returns a boolean mask of the troughs (i.e. 1 when
    the pixel's value is the neighborhood maximum, 0 otherwise)
    """
    # define an connected neighborhood
    # http://www.scipy.org/doc/api_docs/SciPy.ndimage.morphology.html#generate_binary_structure
    neighborhood = morphology.generate_binary_structure(len(arr.shape),2)
    # apply the local minimum filter; all locations of minimum value
    # in their neighborhood are set to 1
    # http://www.scipy.org/doc/api_docs/SciPy.ndimage.filters.html#minimum_filter
    local_max = (filters.maximum_filter(arr, footprint=neighborhood)==arr)
    # local_min is a mask that contains the peaks we are
    # looking for, but also the background.
    # In order to isolate the peaks we must remove the background from the mask.
    #
    # we create the mask of the background
    background = (arr==0)
    #
    # a little technicality: we must erode the background in order to
    # successfully subtract it from local_min, otherwise a line will
    # appear along the background border (artifact of the local minimum filter)
    # http://www.scipy.org/doc/api_docs/SciPy.ndimage.morphology.html#binary_erosion
    eroded_background = morphology.binary_erosion(
        background, structure=neighborhood, border_value=1)
    #
    # we obtain the final mask, containing only peaks,
    # by removing the background from the local_min mask
    detected_maxima = local_max ^ eroded_background
    return np.where(detected_maxima)
```

23
```
def plot_potentials():
    global dest
    TotPot = np.zeros((Nx,Ny))
    TotPot =- np.sqrt((dest[0]-x[:,None])**2+(dest[1]-y[None,:])**2)
    TotPot += 0.003*TrailPotential
    maxima=detect_local_maxima(TotPot)
    cs = plt.contourf(X, Y, TotPot, levels=np.linspace(TotPot.min(),TotPot.max(),1000),cmap='PuBu_r
    cbar = plt.colorbar()
    print(maxima)
    plt.scatter(x[maxima[0]],y[maxima[1]])
```

```
    plt.show()
    # commit test
```

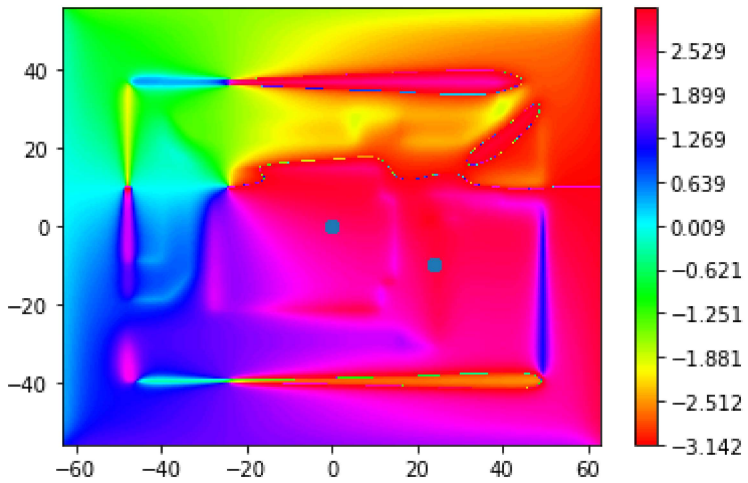
24
```
plot_potentials()
```

```
(array([ 58, 156, 156, 156], dtype=int64), array([262,  62, 262, 373], dtype=int64))
```



25
```
#Plot the direction
scgrad=np.arctan2(grad[1],grad[0])
levels = np.linspace(-np.pi, np.pi, 360)
cs = plt.contourf(X, Y,scgrad, levels=levels,cmap='hsv')

cbar = plt.colorbar()
plt.scatter(track[0:1999,0],track[0:1999,1])
#plt.scatter(start, dest)
print(start)
print(dest)
plt.show()
```

```
[24.   -9.75]
[-24.     9.75]
```



*
```
plot_path()
```

*
```
#Integrate z, trapezoid rule eq 20
# def calc_tr():
#     global xi,yi,TrailPotential,expdist,z,Weight,Nx,Ny
#     for xi in range(0,Nx):
#         for yi in range(0,Ny):
#             TrailPotential[xi,yi]=np.sum(expdist[Nx-1-xi:2*Nx-1-xi,Ny-1-yi:2*Ny-1-yi]*z[:,:]*Weigh
```